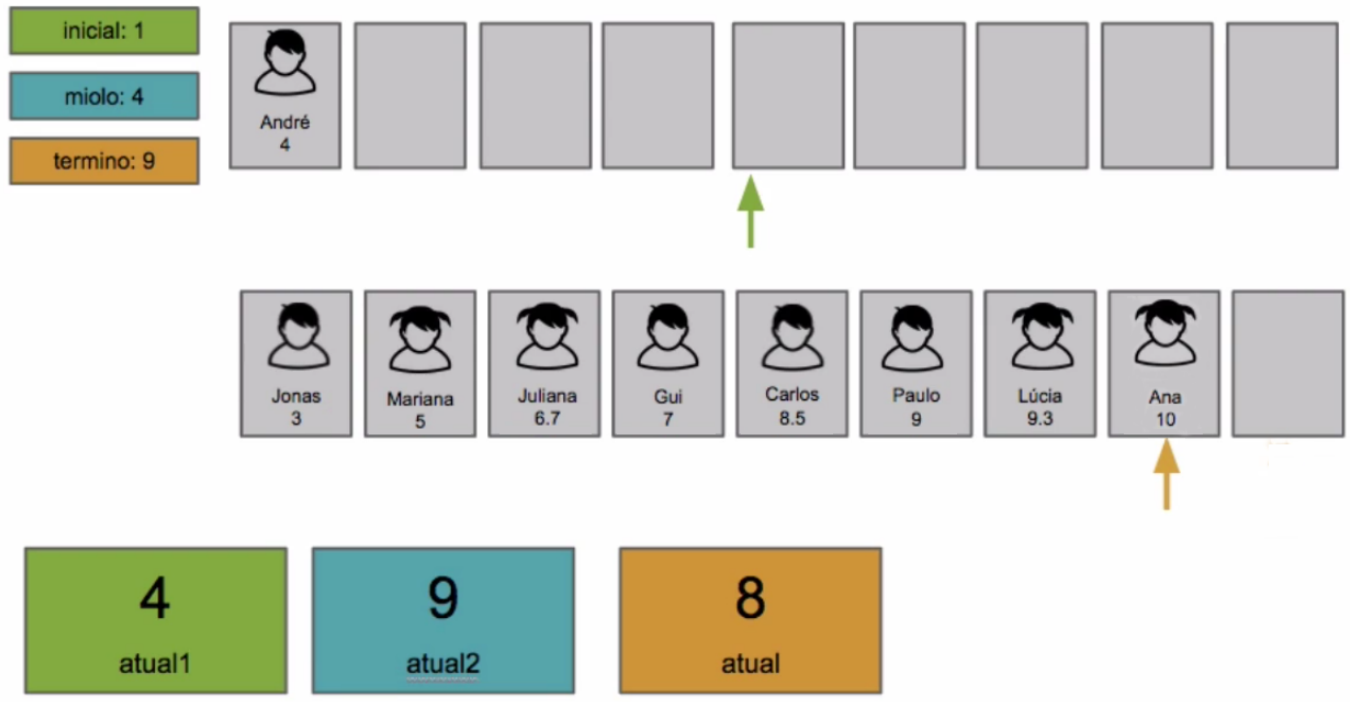
03

Ajustando as pontas em um único array

Transcrição

Quando terminei o algoritmo, com o `inicial` igual a 1, ele não funcionou bem. Por quê? Porque ele criou um *array* de tamanho 9, sendo que ele precisava apenas de espaço para 8. Ele nos devolveu esse *array*:

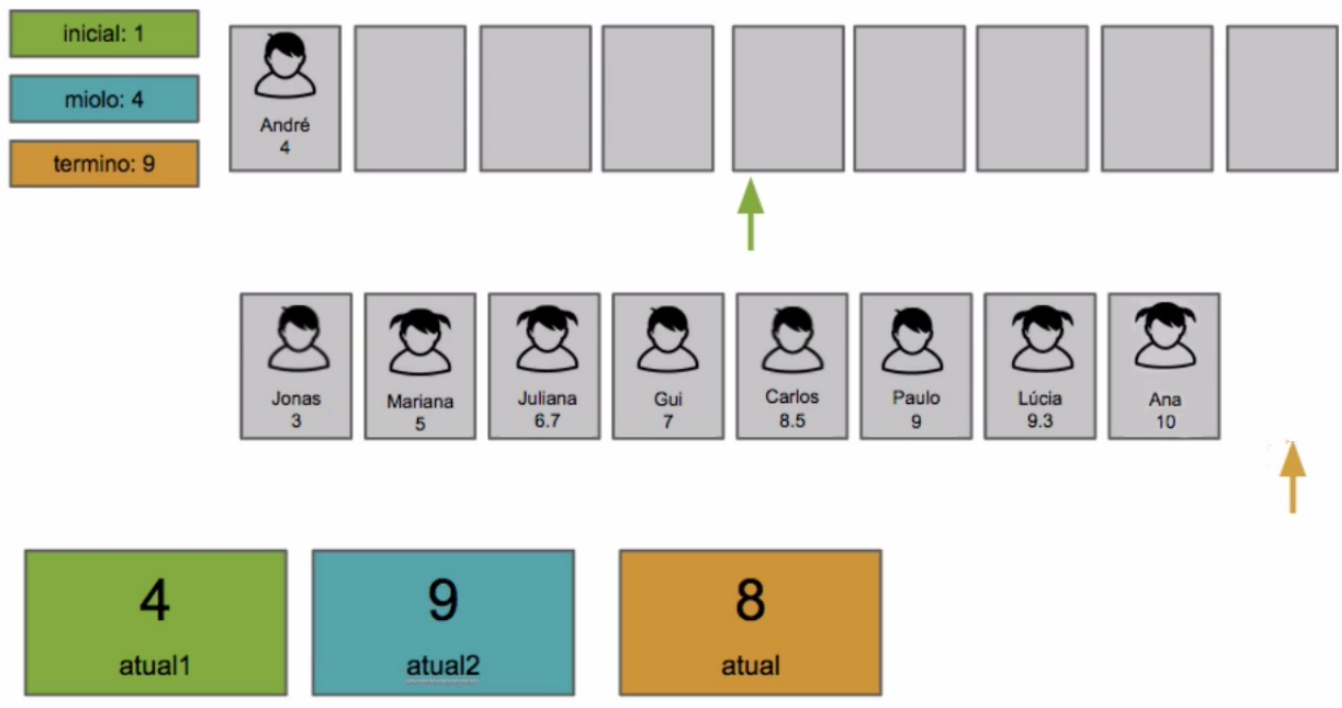


O que nós queríamos é que o primeiro elemento do *array* da esquerda (o Jonas) permanecesse lá, e que os outros elementos estivessem dispostos no novo array*.

E se modificássemos também o valor do `termino` ? Se ele tivesse que terminar antes e sobrasse algum elemento, ele também deveria permanecer na mesma posição. No final, o que restasse, deveria de fato continuar. Pedimos que a carta do coringa continuasse no monte e que a prova do aluno trapaceiro permanecesse no começo da pilha e do melhor aluno, no fim. Queríamos que um determinado elemento permanecesse no início e que outro ficasse no fim. E o algoritmo nos devolveu um *array* que terminou com um espaço vazio. Isto significa que receberíamos a mensagem de erro *NullPointerException*.

Para resolver o problema, primeiramente precisamos evitar criar coisas que não precisamos. Se precisaremos de 8 elementos, por que criamos um *array* para 9? Foi o primeiro erro cometido. Antes ele tinha um tamanho 9, porque nós tínhamos 9 elementos. Quantos elementos temos agora? Apenas 8. Este número surgiu do valor do `termino` , que é igual a 9, menos o `inicial` , que é igual a 1. É assim que sabemos quantos elementos temos no meio e iremos utilizar.

Então, no momento de criar um novo *array* não será de tamanho 9, mas de 8. Será o suficiente e não teremos *NullPointerException*.



Corrigimos o erro de colocar um elemento a mais no *array*, o que faremos agora? Vamos colocar os oito elementos de volta ao *array* em que estavam. E o nosso problema estará resolvido!

Para isto, vamos criar um novo `for` que passe por todo o *array* de resultado e que copie cada elemento para o *array* de onde saiu. Também precisaremos utilizar uma variável que passe por todos os elementos, que irá de 0 até 8.

Como fazemos para copiar os itens para o outro *array*? Podemos selecionar o elemento 0 e inserir na posição 0? Não irá funcionar. O melhor seria colocá-lo na posição seguinte a inicial, a 1. Então, o elemento que está na posição 0 vou movê-lo para a posição inicial + 0, que será a 1. O elemento na posição 2, vamos colocá-lo na posição inicial + 2, que será a posição 2. Seguimos repetindo o mesmo processo com os demais elementos, até termos movimentos todos os itens para as posições adequadas do outro *array*. O nosso *array* original ficou corretamente ordenado.

O que precisamos fazer em seguida? Iremos remover os indicadores que não iremos utilizar, criaremos um *array* de tamanho `termino` menos `inicial` , e após intercalar os elementos que fazem parte, basta copia-los de volta para a origem. No entanto, para conseguirmos fazer isto, precisamos sempre adicionar a posição `inicial` para deslocá-los adequadamente.

Copiando parte do *array* em Java

Vamos tentar uma variação do intercala? Já que recebemos os parâmetros de `inicio` , o 'miolo', e o 'termino', iremos determinar que ele ignore o primeiro elemento e que este permaneça na mesma posição.

```
Nota[] rank = intercala(notas, 1, 4, notas.length);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
    }
}
```

Vamos tentar rodar o programa? O programa irá nos mostrar as notas :

```
jonas 3.0
mariana 5.0
```

```
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
lucia 9.3
ana 10.0

Exception in thread "main" java.lang.NullPointerException
        at br.com.alura.notas.TestaIntercalaEmUmArray
```

Irá aparecer uma mensagem de erro `NullPointerException`. O algoritmo quebrou o programa quando mandamos ele começar a partir da posição da Mariana! Vamos verificar o nosso código e entender o que aconteceu?

```
Nota[] rank = intercala(notas, 1, 4, notas.length);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
    }
```

Nós vimos que o `intercala` está recebendo os valores 1, 4 e 9. Qual será o tamanho do *array* que ele irá criar? Será o mesmo tamanho do *array* original, 9:

```
private static Nota[] notas, int inicial, int miolo, int termino) {
    Nota[] resultado = new Nota[notas.length];
}
```

Logo, o `resultado` terá 9 casinhas.

Após executar o código completo, quantos elementos foram copiados? Faremos um `System.out.println` (depois dos *whiles*) e veremos quanto vale o `atual`:

```
System.out.println(atual);

return resultado;
```

Ao rodarmos o programa, veremos que ele só copiou 8 elementos:

```
8
jonas 3.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
lucia 9.3
ana 10.0

Exception in thread "main" java.lang.NullPointerException
        at br.com.alura.notas.TestaIntercalaEmUmArray
```

Por que ele não copiou todos os elementos? Porque pedimos para começar a partir do 'inicial':

```
int atual1 = inicial;
```

Então, ele ignorou o primeiro elemento e o deixou de fora. Nós ficamos com um *array* que tem na verdade um tamanho 9, mas que na realidade só tem 8 elementos, porque pedimos que ele começasse pelo 1. Observe, que na verdade, quando formos criar o nosso *array* `Nota[notas.length]`, não queremos que ele tenha tamanho 9. É suficiente que ele seja tamanho 8. Podemos generalizar o tamanho para `termino - inicial`.

```
private static Nota[] intercala(Nota[] notas, int inicial, int miolo, int termino) {
    Nota[] resultado = new Nota[termino - inicial];
```

Isto já resolve parte do problema. O programa já irá imprimir os 8 elementos intercalados corretamente:

```
8
jonas 3.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
lucia 9.3
ana 10.0
```

No entanto, onde está o primeiro elemento que deveria estar no começo que nós pedimos que ele ignorasse. Por que ele ficou de fora? Porque nós sempre estamos devolvendo um *array* novo e não é isto o que queremos. Queremos que ele devolva o mesmo *array*, com o primeiro elemento. Apenas precisamos que os outros elementos sejam intercalados. Este é o nosso objetivo: intercalar os elementos que vêm depois do primeiro. Para conseguirmos fazer isto, temos que parar de retornar o mesmo `resultado`. Ele imprime apenas os 8 elementos que intercalamos. Ele tem os itens ordenados na ordem certa, que estão entre o `inicial` e o `termino` e o resto permanece no `notas`. O que vamos fazer? Vamos copiar os elementos do `resultado` e movê-los para o `notas`.

Criaremos um `for`, no fim do código, que irá passar por cada um dos elementos que copiamos:

```
for(int contador = 0; contador < atual ; contador++) {
    }
```

Por que o `contador` é menor que `atual`? Porque `atual` é o número de elementos que nós intercalamos. Se intercalamos 8 elementos, iremos querer que todos estejam no nosso *array* original.

Isto é: nós iremos selecionar o primeiro elemento do `resultado[contador]` e movê-lo para o `notas`, na a posição que ele pertence (`inicial + contador`).

```
notas[inicial + contador] = resultado[contador];
```

Se o `contador` é igual a 0 e o `inicial` é igual a 1, então o elemento pertencerá à posição 1. Quando o `contador` for igual a 1 e o `inicial` for 1, ele pertencerá à posição 2. Será o mesmo até terminarmos a intercalação. No fim, apenas teremos que retornar as notas: `return notas;`

```
for(int contador = 0; contador < atual ; contador++) {
    notas[inicial + contador] = resultado[contador];
}
return notas;
```

Iremos testar se copiamos corretamente os elementos do `resultado` para dentro do `notas`. O programa irá imprimir:

```
8
andre 4.0
jonas 3.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
lucia 9.3
ana 10.0
```

Ele respondeu que intercalou 8 elementos, deixou o primeiro no lugar e ordenou os seguintes.

Antes, quando nós devolvíamos o *array*, como o valor do `inicial` modificado, o algoritmo não funcionava. A razão é que estávamos criando um *array* maior do que precisávamos. Depois, criamos um *array* do tamanho adequado, que seguia até o `termino` menos o `inicial`. É o tamanho que de fator gostaríamos de analisar.

Fizemos todo o processo, no fim, o nosso `for` pedia para copiar de volta para o `notas` todos os elementos que foram intercalados.

Ao rodarmos de novo o algoritmo, teremos o resultado correto.

Agora podemos remover a linha do `System.out` que informa a quantidade de elementos intercalados :

```
System.out.println(atual);
```

E o nosso *array* estará ordenado:

```
- andre 4.0
- jonas 3.0
- mariana 5.0
- juliana 6.7
- guilherme 7.0
- carlos 8.5
- paulo 9.0
- lucia 9.3
- ana 10.0
```

Podemos mandar intercalar apenas o trecho que queremos. Estamos intercalando a partir do primeiro elemento, mas ainda temos a opção de intercalar todos os itens. Basta alterar a posição `inicial`.

```
Nota[] rank = intercala(notas, 0, 4, notas.length);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
    }
}
```

Se testarmos, veremos que todos os elementos serão intercalados.

```
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
lucia 9.3
ana 10.0
```

Podemos intercalar também todos os elementos, menos o último. Basta subtrair -1 do `notas.length` .

```
Nota[] rank = intercala(notas, 0, 4, notas.length-1);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
    }
}
```

Se imprimirmos o resultado, a Ana continuará no fim da lista de elementos. Afinal, ela teve a maior pontuação.

```
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
lucia 9.3
ana 10.0
```

Vamos testar se subtraíssemos -3 do `notas.length` :

```
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
carlos 8.5
paulo 9.0
guilherme 7.0
lucia 9.3
ana 10.0
```

O Guilherme, a Ana e a Lúcia permaneceram na posição, enquanto os outros elementos foram intercalados.

Agora a nossa função `intercala()` intercala adequadamente o trecho determinado.